パターン認識　レポート 1

1019163　日置竜輔 *

2021 年 7 月 20 日

目次

* 公立はこだて未来大学　システム情報科学部　複雑系知能学科　複雑系コース　B3

1　はじめに

今回のレポートの最終目標は顔画像に対して、DCT（Discrete Cosine Transformation）係数を求めることであるが、これを行うに当たって、画像処理の知識や DCT の求め方などを知らなかったため、まずはこれらの勉強から始めた。

1.1　画像処理の基本について学習する

先ほど述べたようにまずは画像処理をコーディングするための基礎知識として画像処理 100 本ノック!!というもので知識を深めた。今回はモジュールが豊富という観点から Python を使用することにした。
ここでは、主に画像処理を行うために必須となる matplotlib, opencv, numpy, skimage などの使い方を勉強した。このノックを行ったことによってある程度画像処理の行い方を学習することができた。

1.2　DCT とは何か

画像処理の基本が分かった所で次は課題となっている DCT（以下、離散コサイン変換と呼ぶ）について具体的に調べた。
離散コサイン変換とは、離散信号を周波数領域へ変換する方法の一つであり、信号圧縮に広く用いられている。
離散コサイン変換は、有限数列を、余弦関数数列 $\cos(nk)$ を基底とする一次結合（つまり、適切な周波数と振幅のコサインカーブの和）の係数に変換する。余弦関数は実数に対しては実数を返すので、実数列に対しては DCT 係数も実数列となる。
離散コサイン変換とよく区別されるのが離散フーリエ変換（DFT: discrete Fourier transform）である。離散フーリエ変換は、実数に対しても複素数を返す $\exp(ink)$ を使用するが、DCT は実数であれば、複素数ではなく実数で返すのが大きな特徴となっている。
離散コサイン変換では、係数が実数になる上、特定の成分への集中度があがるため、JPEG などの画像圧縮、AAC や MP3、ATRAC といった音声圧縮、デジタルフィルタ等広い範囲で用いられている。

1.3 DCT を数学的に理解する

離散コサイン変換の数学的な具体的な定義は以下のように定められている。

$$0 \leqq u, v < T$$

$$F(u,v) = \frac{2}{T} C(u)C(v) \sum_{y=0}^{T-1} \sum_{x=0}^{T-1} I(x,y) (\cos \frac{(2x+1)u\pi}{2T}) \cos(\frac{(2y+1)v\pi}{2T}) \tag{1.3.1}$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & u = 0 \\ 1 & u \neq 0 \end{cases}$$

このような定義が分かった所で次は実際に Python でコーディングを行う。

2　プログラムによる実装

顔画像の DCT 係数を計算するに当たって、今回は女優として活躍されている浜辺美波さんの顔画像の DCT 係数を求めることにした。

図 1 今回使用する浜辺美波の顔画像

2.1　python による実装

Listing 1　DCT 係数を求めるソースコード

```
# DCT hyoer-parameter
img_128 = io.imread("img/hamabe-minai.png")

# DCT
def dct(img, T=8, channel=3):
    H, W, _ = img.shape
    F = np.zeros((H, W, channel), dtype=np.float32)

    theta = np.pi / (2 * T)

    for c in range(channel):
        for vi in range(0, H, T):
            for ui in range(0, W, T):
                for v in range(T):
                    for u in range(T):
                        cu = 1 / np.sqrt(2) if u == 0 else 1
                        cv = 1 / np.sqrt(2) if v == 0 else 1
                        coef1, coef2 = np.meshgrid(np.cos((2 * np.arange(0, T) + 1) *
                            u * theta), np.cos((2 * np.arange(0, T) + 1) * v * theta))
                        F[vi + v, ui + u, c] = 2 * cu * cv * np.sum(img[vi : vi + T,
                            ui : ui + T, c] * coef1 * coef2) / T
```

```
20      return F
21
22  # DCT
23  F = dct(img_128) # 画像に対して係数を求める DCT
```

2.2 DCT 係数を出力する

2.1 で求めた F に 128*128 画素の 3 次元ベクトルが格納されているので、これを出力することで DCT 係数を得ることができる。

Listing 2　DCT 係数を出力するソースコード

```
1  with open('output.txt', 'w') as f: # という名前のテキストファイルを作成する output
2      for y in range(len(F)): # 縦の画素 128
3          for x in range(len(F[y])): # 横の画素 128
4              f.write(str(y) + " " + str(x) + "\n") # (y, x) の座標を書き込む
5              f.write(str(F[y][x]) + "n") # ３次元ベクトルを書き込む
```

上記のコードで得られた係数のサンプルが下図である。

２つの数字は２次元ユークリッド座標 (y, x) を表しており、続く３つの配列は成分のベクトルを表している。.txt の拡張子で保存するには str 型で保存する必要があるので、list で受け取った配列を str に戻して入力する実装を行う。

```
0 0
[1892.375 1933.75  1981.875]
0 1
[ 0.45859027 -1.6857488  -1.114271  ]
0 2
[ 0.10727768  1.847759   -0.12369221]
0 3
[ 0.80885845 -0.15200073 -0.81979185]
0 4
[-0.125 -1.25  -0.375]
0 5
[-0.5741629  -0.01349872  0.62912875]
0 6
[-0.720931   -0.76536685 -0.6252601 ]
0 7
[-0.49323413  0.09982145  0.05512228]
0 8
[1895.125 1941.75  1995.75 ]
0 9
[-2.652249  -3.9256148 -6.350971 ]
0 10
[ 0.2986194  -0.69290966 -1.8641735 ]
0 11
[0.9165733  0.18690145 0.43445176]
0 12
[1.125 1.    1.5  ]
0 13
[0.18721966 0.3632847  0.01541255]
0 14
[-0.25899124  0.28701258 -0.58082426]
0 15
[-0.16708522 -0.2182667  -0.7753579 ]
0 16
[1907.875 1946.25  1991.   ]
0 17
[-9.813034  -3.4166424  1.4274864]
0 18
[0.6252601 2.039101  4.0385733]
0 19
[2.2923067 1.3215312 1.8500164]
0 20
[ 0.375 -0.25  -0.75 ]
0 21
[-0.08112495 -0.6052338  -1.2163999 ]
0 22
[-0.12369221 -0.3034271  -0.04924357]
0 23
[-0.0748765   0.58206725  1.0886198 ]
```

図 2 DCT 係数の出力結果の一部分

3　逆離散コサイン変換を行う

第 2 章で離散コサイン変換を行い、DCT 係数を求める作業を行った。そして、課題とはなっていないが、これらの数値から元の画像に復元できないかと思い、実装してみることにした。

3.1　逆離散コサイン変換

逆離散コサイン変換（IDCT: Inverse Discrete Cosine Transformation）とは離散コサイン変換の逆（復号）であり、次式で定義される。ここでいう K は復元時にどれだけ解像度を良くするかを決定するパラメータである。

K = T の時は、DCT 係数を全部使うので IDCT 後の解像度は最大になるが、K が 1 や 2 などの時は復元に使う情報量（DCT 係数）が減るので解像度が下がる。これを適度に設定することで、画像の容量を減らすこ

とができる。

IDCT は数学的には以下のように定義されている。

$$1 \leqq K \leqq T$$

$$F(x,y) = \frac{2}{T} \sum_{y=0}^{T-1} \sum_{x=0}^{T-1} C(u)C(v)F(u,v)(\cos\frac{(2x+1)u\pi}{2T})\cos(\frac{(2y+1)v\pi}{2T}) \tag{3.1.1}$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & u = 0 \\ 1 & u \neq 0 \end{cases}$$

上記のように定められているため、これを用いて画像の復元を行う。

3.2 IDCT を用いて顔画像の復元

コードは下記のようなものになった。

Listing 3　DCT 係数を元に画像を復元するソースコード

```
# DCT hyoer-parameter
img_128 = io.imread("img/hamabe-minai.png")

# DCT
def dct(img, T=8, channel=3):
    H, W, _ = img.shape
    F = np.zeros((H, W, channel), dtype=np.float32)

    theta = np.pi / (2 * T)

    for c in range(channel):
        for vi in range(0, H, T):
            for ui in range(0, W, T):
                for v in range(T):
                    for u in range(T):
                        cu = 1 / np.sqrt(2) if u == 0 else 1
                        cv = 1 / np.sqrt(2) if v == 0 else 1
                        coef1, coef2 = np.meshgrid(np.cos((2 * np.arange(0, T) + 1) *
                            u * theta), np.cos((2 * np.arange(0, T) + 1) * v * theta))
                        F[vi + v, ui + u, c] = 2 * cu * cv * np.sum(img[vi : vi + T,
                            ui : ui + T, c] * coef1 * coef2) / T
```

```
    return F

# IDCT
def idct(F, T=8, K=8, channel=3):
    H, W, _ = F.shape
    out = np.zeros((H, W, channel), dtype=np.float32)

    theta = np.pi / (2 * T)

    c_mat = np.ones([T, T])
    c_mat[0] /= np.sqrt(2)
    c_mat[:, 0] /= np.sqrt(2)
   # print(c_mat)
    for c in range(channel):
        for yi in range(0, H, T):
            for xi in range(0, W, T):
                for y in range(T):
                    for x in range(T):
                        coef1, coef2 = np.meshgrid(np.cos((2 * x + 1) * np.arange(0, T
                            ) * theta), np.cos((2 * y + 1) * np.arange(0, T) * theta))
                        out[yi + y, xi + x, c] = 2 * np.sum(F[yi : yi + K, xi : xi + K
                            , c] * coef1[:K, :K] * coef2[:K, :K] * c_mat[:K, :K]) / T

    out = np.clip(out, 0, 255)
    out = np.round(out).astype(np.uint8)

    return out

# DCT
F = dct(img_128) # 係数を求める DCT

# IDCT
out = idct(F) # 係数を用いて顔画像を復元する DCT

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.title("Original")
plt.imshow(img_128)

plt.subplot(1, 2 ,2)
plt.title("Face image after using IDCT")
plt.imshow(out)
plt.show()
```

上記のコードを実行して得られた結果が以下のようになる。

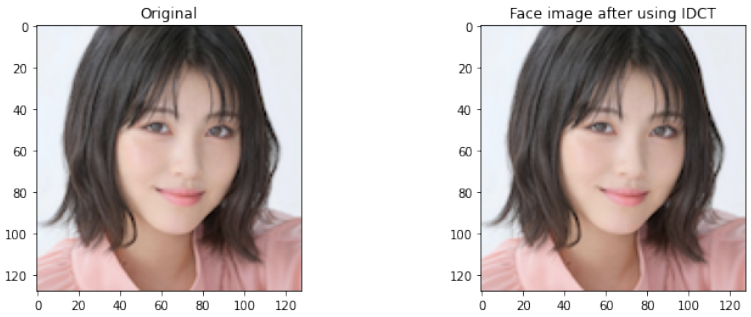

図 3 DCT 係数を用いて復元した後の浜辺美波の顔画像

得られた画像の方が色合いが少し濃くなっていることがわかるが、顔の形や色はかなり再現されていることが確認できる。

4　最後に

今回は指定された顔画像の DCT 係数を求めるということだけでなく、さらに IDCT を用いて顔画像の復元を行うということまで実装した。結果的にほとんど元の画像と一致して結果が得られたので安心した。

DCT は前述でも述べたように、JPEG などの画像圧縮、AAC や MP3、ATRAC といった音声圧縮、デジタルフィルタ等広い範囲で用いられており、画像処理についてはまだまだ知らないことが多くあると感じたので、講義が終わった後も自主的に学んでいきたいと感じた。